Abstract Constraints: A General Framework for Solver-Independent Reasoning on Product Line Models

Raul Mazo, Camille Salinesi, Daniel Diaz

Abstract Constraints: A General Framework for Solver-Independent Reasoning on Product Line Models

Raul Mazo[1,2], Camille Salinesi[1], Daniel Diaz[1].
[1] CRI, Panthéon Sorbonne University, 90, rue de Tolbiac, 75013 Paris, France
[2] Departamento de Ingeniería de Sistemas, Universidad de Antioquia, Medellín, Colombia
raulmazo@gmail.com, {camille.salinesi, daniel.diaz}@univ-paris1.fr

Product Line Engineering (PLE) is a paradigm for reuse-based complex systems development that is well installed in the industry. Among the proven benefits are reduced time to market, better asset reuse, and improved software quality [1]. To be successful, PLE must efficiently manage the variability — the capacity of product line's artifacts to vary — present in the products that form a Product Line (PL). Several modeling approaches have been proposed to represent the artifacts of a PL, their properties and relationships. All these notations can be used to describe in a single Product Line Model (PLM) all the legal combinations of features (qualities, artifacts, etc) [2]. In this context, being able to reason about the PLM is an important success factor in the PLE strategy. Reasoning on PLMs is achieved by querying the models in order to verify, analyze or configure them [3]. For instance, PLMs can be verified to guarantee that they do not have undesirable properties affecting the correctness of the products they help develop. Several approaches are available in the literature to support automatic reasoning on PLMs. Several approaches consist in transforming the PLMs into a constraint program that can be executed by a solver. For example, Satisfiability (SAT) solvers are used to analyze PLMs specified as Boolean constraints. Others use SAT or constraint over finite domains solvers to find the number of solutions that can be configured on a PLM. Interestingly, it is actually well know that for this task Binary Decision Diagram (BDD) solvers are more efficient. Thus, authors seem to undermine the efficiency of certain reasoning operations to prioritize others. One reason might be that the transformation is guided by the solver to be used and not by nature of the PLMs or the efficiency/limitations of using one solver or another one.

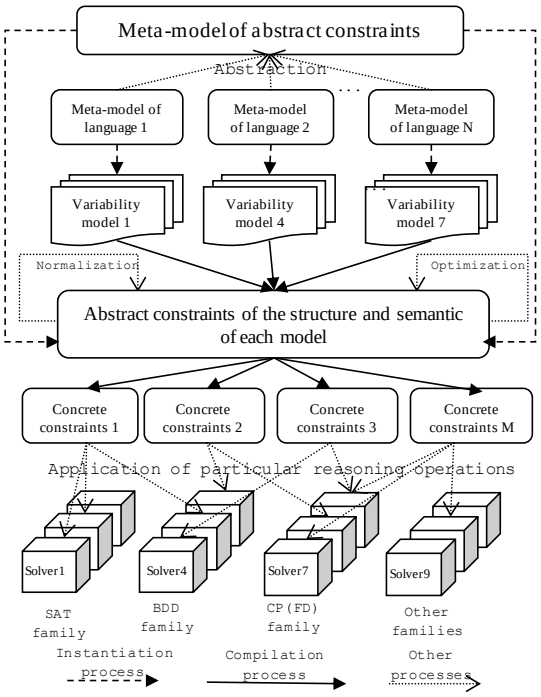

Figure 1: Constraint based configuration overview

To overcome these limitations, we propose to represent the semantics of PLMs as abstract constraints with a unique notation that encompass other constraint languages (e.g., over Booleans, Integers, Reals, trees, lists, etc.). As Figure 1 sows it, once a PLM is specified as abstract constraints, it can be compiled with the platform in any constraint language depending on the analysis to achieve and on solver to use for the analysis.

In order to do that, our first concern is to define a notation that consists in a constraints system allowing represent product lines. According to Saraswat [4], a constraint system can be defined as a tuple $(D, \vdash)$ where D is a set of first-order formulas closed under conjunction and existential quantification, $\vdash$ is an entailment relation between a finite set of formulas (taken from D) and a single formula and $\vdash$ must be generic (that is: $S[t/X] \vdash d[t/X]$ whenever $S \vdash d$, for any term t). A constraints system for representing product lines over a parameterizable domain X (e.g., X=Finite Domain, X=Reals, X=Booleans), is a tuple of the minimal set of first-order formulas allowing to represent product lines. For us, the minimal collection of complete variability constraints to represent a product line is {*mandatory*, *optional*, *requires*, *excludes*}, but others can be added, and an entailment relation between these constraints can be defined. The entailment relation is given by rules. We can therefore define a kind of operational semantic of entailment between constraints adapted to the domain of the solver on which the constraints system will be executed. So, these rules can be reduced to conjunction operators between complete variability constrains on PL domain. It is simply because any product to be configured from the product line representation must satisfy all the constraints of the PL which implies entail the complete variability constraints (we are talking about the *mandatory*, *optional*, *requires*, *excludes* and other complete constraints and not about the atomic constrains in them) by means of conjunctions.

The first-order formulas representing the variability constraints of a product line are:
mandatory: $\exists a, b(Variable(a), Variable(b) \wedge (a \Rightarrow b) \Leftrightarrow (b \Rightarrow a))$
optional: $\exists a, b(Variable(a), Variable(b) \wedge ((a \Rightarrow b) \vee (b \Rightarrow a))$
requires: $\exists a, b(Variable(a), Variable(b) \wedge (a \Rightarrow b))$
excludes: $\exists a, b(Variable(a), Variable(b) \wedge (a \oplus b))$

Where *Variable(x)* means that *x* is a variable in a non-specified domain. Now, our next issue is to identify a proper form for the components that allows transforming constraints specified with the generic notation into some kind of constraints in a particular domain, and the other way round. In order to achieve this, we are developing a series of transducers. The difficulty in developping these is that they must be monotonic and continuous in the orderinginformation . Because of the first-order structure of the constraints, we require that the transducers be generic in all the variables. To be generic in a variable *V*, means that if the transducer can produce the information *d* on input *c*, then it can also produce the information *d[t/V]* for input *c[t/V]* for any *t*.

In the context of PLMs, the design of these transducers depends of the target back-end solvers than shall be used to achieve the PLM analyses The details are not provided in this paper for the sake of space, but examples are given in [3,5].

References

[1] P. Clements and L. M. Northrop. Software Product Lines - Practices and Patterns. Addison-Wesley, 2001.
[2] Salinesi, C., Mazo, R., Diaz, D., Djebbi, O. Solving Integer Constraint in Reuse Based Requirements Engineering. 18th IEEE International Conference on Requirements Engineering (RE'10). Australia, 2010.
[3] Salinesi C., Mazo R., Djebbi O., Diaz D., Lora-Michiels A. Constraints: the Core of Product Line Engineering. 5th IEEE International Conference on Research Challenges in Information Science. France, 2011.
[4] V. Saraswat. The Category of Constraint Systems is Cartesian-Closed. In Logic In Computer Science, IEEE Press 1992.
[5] Mazo R., Salinesi C., Diaz D., Lora-Michiels A. Transforming Attribute and Clone-Enabled Feature Models Into Constraint Programs Over Finite Domains. 6th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), Springer Press, Beijing–China, 8-11 June 2011.